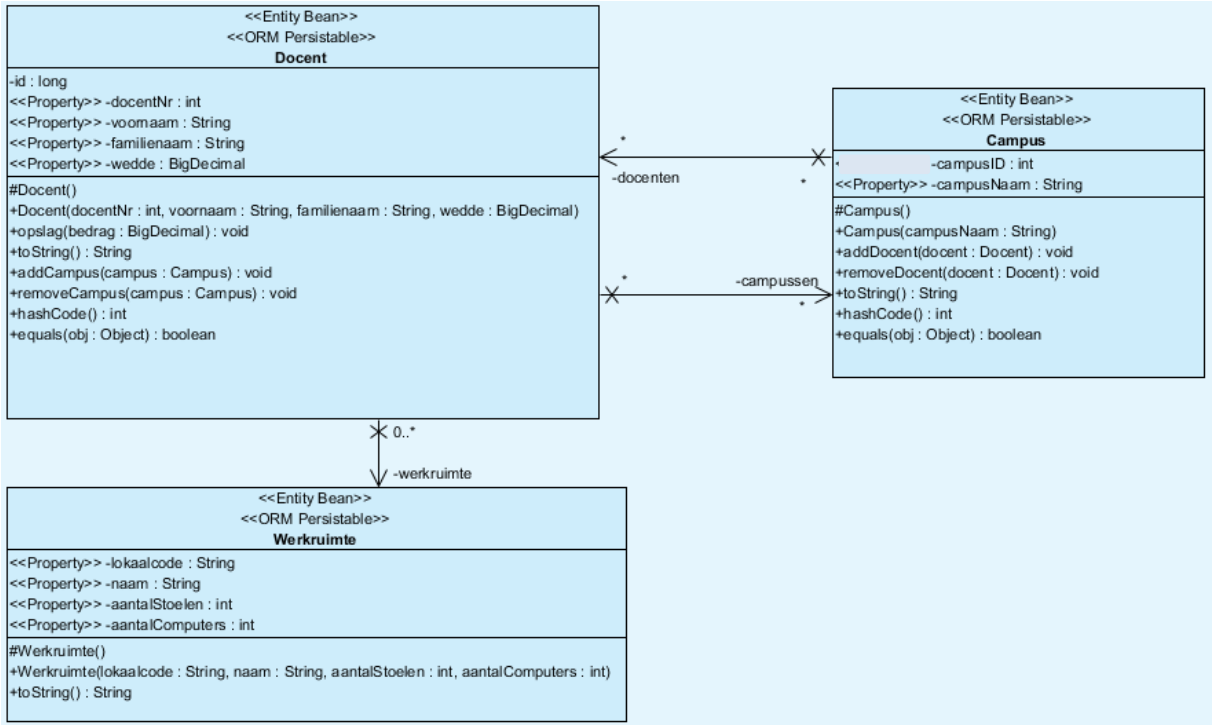
2. JPA-ORM mapping.

2.1. Uitbreiding domein.

We breiden ons domein uit. De docenten kunnen aan meerdere campussen verbonden zijn. Bij een campus horen meerdere docenten. Elke docent heeft slechts één werkruimte (kantoor). Een werkruimte wordt door meerdere docenten gedeeld.

In het ontwerp voorzien we een bidirectionele associatie tussen docent en campus en een unidirectionele associatie tussen docent en werkplaats.



Voeg de nieuwe klassen en relaties toe. Maak van Campus en Werkruimte ook entity klassen. We hergebruiken de naamgeving in het domein voor de namen in de DB.

We starten met een nieuwe DB, school2DB. Creëer de nieuwe lege DB school2DB. Pas de persistence.xml aan:
--> JDBC Connection --> jfbc:derby://localhost:1527/school2DB
--> Vergeet ook niet de nieuwe entity klassen toe te voegen.

Maak een java klasse MAINoef3 die een initieel aantal domein objecten aanmaakt en persistent maakt.

```
Docent a  = new Docent(123, "Jan", "Baard", new BigDecimal(8000));
Docent b  = new Docent(456, "Piet", "Baard", new BigDecimal(10000));
Docent c  = new Docent(789, "Joris", "ZonderBaard", new BigDecimal(12000));
Campus gent = new Campus("Gent");
Campus aalst = new Campus("Aalst");
Werkruimte r1 = new Werkruimte("SCH123", "zolder", 12, 6);
```

Werkruimte r2 = new Werkruimte("SCH555", "kelder", 4, 4);
Werkruimte r3 = new Werkruimte("AA222", "dak", 10, 2);

Na de run bekijk je het resultaat in de DB. Buiten de 3 tabellen (DOCENTEN, CAMPUS, WERKRUIMTE) nodig voor elke entity klasse, zal nu ook een lege tussentabel aangemaakt zijn met de naam DOCENTEN_CAMPUS voor de veel op veel relatie.

#	DOCENTEN_ID	CAMPUSSEN_CAMPUSID

CAMPUS

#	CAMPUSID	CAMPUSNAAM
1	1	Gent
2	2	Aalst

DOCENTEN

#	ID	PERSONEELSNR	FAMILIENAAM	VOORNAAM	WEDDE	WERKRUIMTE_LOKAALCODE
1	1	789	ZonderBaard	Joris	12000	<NULL>
2	2	123	Baard	Jan	8000	<NULL>
3	3	456	Baard	Piet	10000	<NULL>

WERRUIMTE

#	LOKAALCODE	AANTALCOMPUTERS	AANTALSTOELEN	NAAM
1	SCH555	4	4	kelder
2	AA222	2	10	dak
3	SCH123	6	12	zolder

2.2. Vervolledig stap 2.1. door het maken van relaties tussen de initiële objecten.

Maak een java klasse MAINoef4, een kopie van MAINoef3. Vervolledig de code om volgende relaties te leggen.

Leg volgende relaties vast:
Docent Jan, Piet en Joris werken op campus Gent.
Docent Jan en Joris werken op campus Aalst.
Docent Jan heeft als werkruimte SCH123.
Docent Piet heeft als werkruimte SCH123.
Docent Joris heeft als werkruimte AA222.

Door de run van MAINoef3 zijn in de DB school2 de tabellen reeds aangemaakt met de nodige records. We veranderen éénmalig (enkel voor deze run) de Table Generation Strategy van Create naar Drop and Create. Anders zouden er opnieuw extra records aangemaakt worden en ook exceptions op dubbele sleutelwaarden ontstaan.

Na de run bekijken we de DB. We zien: DOCENTEN_CAMPUS

#	DOCENTEN_ID	CAMPUSSEN_CAMPUSID
1	1	1
2	1	2
3	2	1
4	3	1
5	3	2

En DOCENTEN

#	ID	PERSONEELSNR	FAMILIENAAM	VOORNAAM	WEDDE	WERKRUIMTE_LOKAALCODE
1	1	123	Baard	Jan	8000	SCH123
2	2	456	Baard	Piet	10000	SCH123
3	3	789	ZonderBaard	Joris	12000	AA222

2.3. Bekijken van al de persistente domeinobjecten.

Maak een java klasse MAINoef5 die de aanwezige domeinobjecten ophaalt, waarna we de lijsten afdrukken. Vergeet de toString methoden van Docent niet aan te passen, en de toString methoden voor Campus en Werkruimte te voorzien.

Om de lijst van bijhorende domeinobjecten op te halen kan je best gebruik maken van een **namedquery**. Voorbeeld voor de Docent klasse (annotation boven de entity klasse plaatsen):

```
@NamedQueries({
   @NamedQuery(name = "Docent.findAll", query = "SELECT d FROM Docent d")})
```

En het ophalen in MAINoef5:

```
List<Docent> docentList;

TypedQuery<Docent> queryD = entityManager.createNamedQuery("Docent.findAll", Docent.class);
docentList=queryD.getResultList();
```

Doe dit ook voor Campus. Om de objecten af te drukken plaats je vervolgens onderstaande code in MAINoef5 (na de aanroep van de named queries)

```
System.out.println(campusList);
System.out.println(docentList);


Docent d1  = docentList.get(0);
Campus dummy1 = d1.getCampussen().iterator().next();
System.out.printf("%s %s\n", d1, d1.getCampussen());
Docent d2  = docentList.get(1);
System.out.printf("%s %s\n", d2, d2.getCampussen());
Docent d3  = docentList.get(2);
System.out.printf("%s %s\n", d3, d3.getCampussen());
Campus c1 = campusList.get(0);
System.out.printf("%s %s\n", c1, c1.getDocenten());
Campus c2 = campusList.get(1);
Docent dummy2 = c2.getDocenten().iterator().next();
```

System.out.printf("%s %s\n", c2, c2.getDocenten());

Je krijgt dan volgende output:

```
run:
[EL Info]: 2015-09-17 16:36:41.378--ServerSession(173175486)--EclipseLink, version: Eclipse Persistence Services - 2.5.2.v20140319-9ad6abd
[EL Info]: connection: 2015-09-17 16:36:41.558--ServerSession(173175486)--file:/C:/usr/workspacenb_OOPIII_theorie_4/OOPIII_JPA_WC_ORMmappin
[1 Gent, 2 Aalst]
[123 Jan Baard 8000 SCH123 zolder 12 6, 456 Piet Baard 10000 SCH123 zolder 12 6, 789 Joris ZonderBaard 12000 AA222 dak 10 2]
123 Jan Baard 8000 SCH123 zolder 12 6 {[2 Aalst, 1 Gent]}
456 Piet Baard 10000 SCH123 zolder 12 6 {IndirectSet: not instantiated}
789 Joris ZonderBaard 12000 AA222 dak 10 2 {IndirectSet: not instantiated}
1 Gent {IndirectSet: not instantiated}
2 Aalst {[123 Jan Baard 8000 SCH123 zolder 12 6, 789 Joris ZonderBaard 12000 AA222 dak 10 2]}
[EL Info]: connection: 2015-09-17 16:36:41.707--ServerSession(173175486)--file:/C:/usr/workspacenb_OOPIII_theorie_4/OOPIII_JPA_WC_ORMmappin
BUILD SUCCESSFUL (total time: 1 second)
```

Je merkt dat de **set**s van de "veel relaties" nog niet zijn "instantiated". Dit komt omdat hiervoor default lazy-loading wordt toegepast. Pas als je echt toegang wil nemen tot een element van de collectie zullen de bijhorende objecten worden opgehaald.

2.4. Eager-loading forceren.

Pas de annotations even aan (enkel voor deze oefening) zodat je het effect van eager-loading kan zien:
attribuut fetch= FetchType.EAGER, geef je mee bij de relationship mapping annotation, tussen ronde haakjes, ingeval van meerdere argumenten scheiden met komma. Run MAINoef5 opnieuw.

Je krijgt dan volgende output:

```
run:
[EL Info]: 2015-09-17 16:42:11.616--ServerSession(689140691)--EclipseLink, version: Eclipse Persistence Services - 2.5.2.v20140319-9ad6a
[EL Info]: connection: 2015-09-17 16:42:11.779--ServerSession(689140691)--file:/C:/usr/workspacenb_OOPIII_theorie_4/OOPIII_JPA_WC_ORMmap
[1 Gent, 2 Aalst]
[123 Jan Baard 8000 SCH123 zolder 12 6, 456 Piet Baard 10000 SCH123 zolder 12 6, 789 Joris ZonderBaard 12000 AA222 dak 10 2]
123 Jan Baard 8000 SCH123 zolder 12 6 [2 Aalst, 1 Gent]
456 Piet Baard 10000 SCH123 zolder 12 6 [1 Gent]
789 Joris ZonderBaard 12000 AA222 dak 10 2 [2 Aalst, 1 Gent]
1 Gent [123 Jan Baard 8000 SCH123 zolder 12 6, 456 Piet Baard 10000 SCH123 zolder 12 6, 789 Joris ZonderBaard 12000 AA222 dak 10 2]
2 Aalst [123 Jan Baard 8000 SCH123 zolder 12 6, 789 Joris ZonderBaard 12000 AA222 dak 10 2]
[EL Info]: connection: 2015-09-17 16:42:11.879--ServerSession(689140691)--file:/C:/usr/workspacenb_OOPIII_theorie_4/OOPIII_JPA_WC_ORMmap
BUILD SUCCESSFUL (total time: 1 second)
```

2.4. Alle docenten van Campus Gent die ook in Aalst werkzaam zijn krijgen tijdelijk als werkplaats de kelder toegewezen.

Maak een nieuwe klasse MAINoef6 die deze klus realiseert.
Het resultaat in de DB:

#	ID	PERSONEELSNR	FAMILIENAAM	VOORNAAM	WEDDE	WERKRUIMTE_LOKAALCODE
1	1	123	Baard	Jan	8000	SCH555
2	2	456	Baard	Piet	10000	SCH123
3	3	789	ZonderBaard	Joris	12000	SCH555